



ENSA  
ÉCOLE NATIONALE DES SCIENCES  
APPLIQUÉES  
KHOURIBGA



**UNIVERSITÉ SULTAN MOULAY SLIMANE**

École Nationale des Sciences Appliquées de Khouribga  
Département Mathématiques et Informatique

Filière : Génie informatique

---

## TUTORIAL : SERVICES WEB AVEC WSDL

---

Réalisé Par  
**Sabbahi kaoutar**  
**Abouanane soukaina**  
**Elfakir wissal**  
**Naji aya**

**Année Académique : 2024**

# Table des matières

<b>Introduction .....</b>	<b>3</b>
<b>Partie 1 : Génération d'un client SOAP avec Apache CXF.....</b>	<b>3</b>
1. Configuration des Variables d'Environnement pour Apache CXF .....	3
3. Création d'un nouveau projet .....	5
4. Configuration de Maven pour Apache CXF .....	6
5. Génération du Client SOAP avec wsdl2java .....	8
6. Les classes nécessaires générées .....	8
7. Exécution des appels SOAP .....	9
<b>Partie 2 : Génération d'un client SOAP avec JAX-WS .....</b>	<b>10</b>
1. Création d'un nouveau projet .....	10
2. Configuration des dépendances Maven.....	10
2.1. Dépendances JAX-WS et JAXB .....	10
2.2. Plugin jaxws-maven-plugin.....	12
3. Commande Maven pour la génération des classes.....	12
4. Les classes nécessaires générées .....	13
5. Création d'une classe client SOAP.....	13
6. Les résultats obtenus .....	14
<b>Conclusion.....</b>	<b>15</b>

# Introduction

Dans ce tuto, nous explorons deux approches pour créer un client SOAP permettant de consommer un service web défini par un fichier WSDL : **Apache CXF** et **JAX-WS**. Ces deux méthodes, populaires en développement Java pour interagir avec des services SOAP, sont présentées ici étape par étape. Chaque partie couvre la configuration de l'environnement, la génération du client et l'exécution des appels SOAP, permettant ainsi de comparer les avantages et particularités de chacune.

## Partie 1 : Génération d'un client SOAP avec Apache CXF

### 1. Configuration des Variables d'Environnement pour Apache CXF

Avant de configurer Maven pour utiliser Apache CXF, vous devez ajouter le chemin d'accès au dossier bin d'Apache CXF dans vos variables d'environnement. Cela permettra d'accéder aux outils CXF directement depuis la ligne de commande.

Étapes :

#### 1. Télécharger Apache CXF :

- Rendez-vous sur le site officiel [d'Apache CXF](#).
- Téléchargez la dernière version stable sous forme de fichier .zip ou .tar.gz en fonction de votre système d'exploitation.

#### 2. Décompresser le fichier :

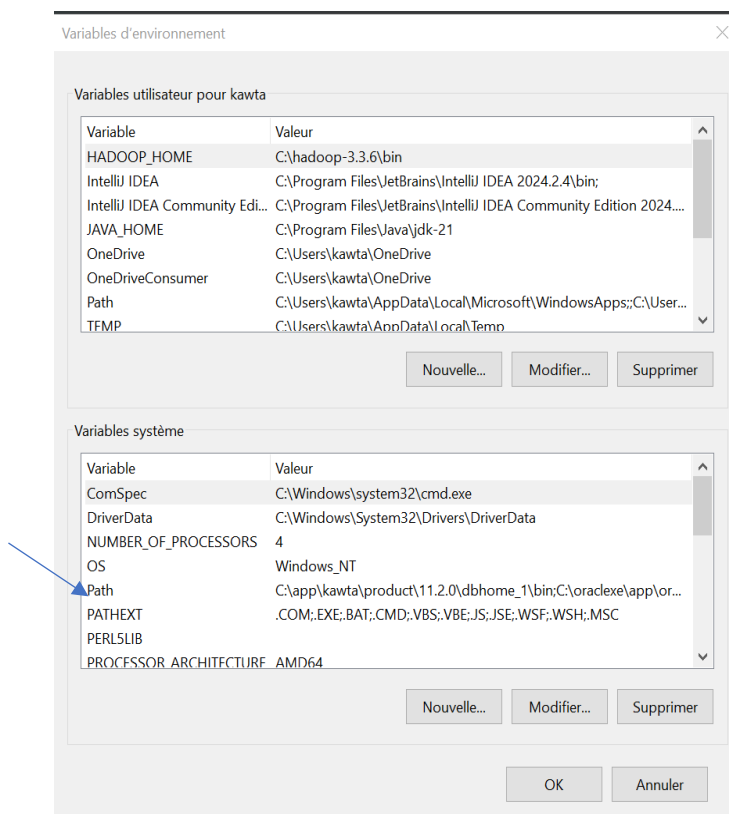
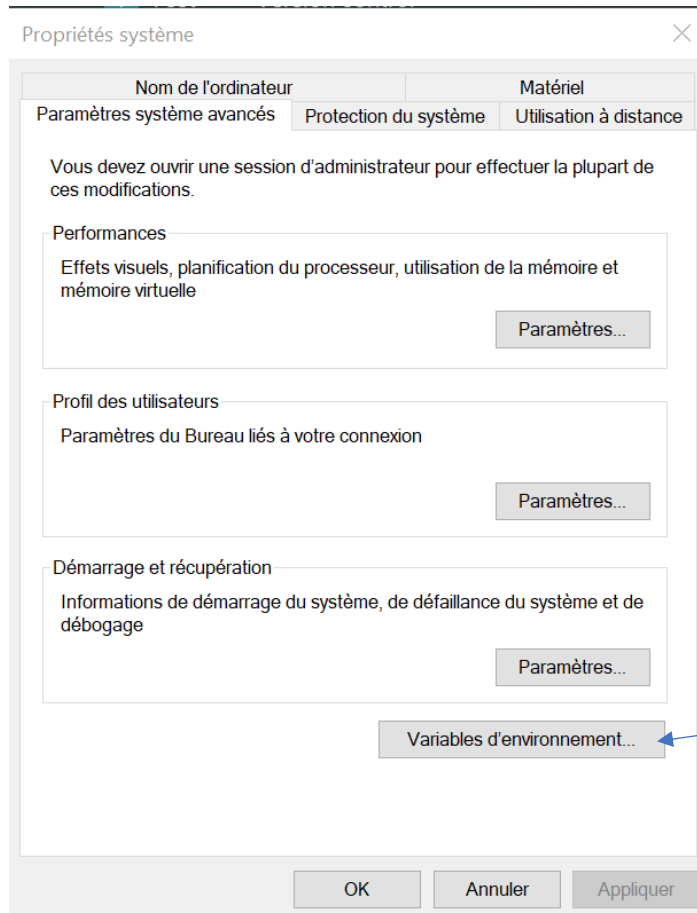
- Extrayez le contenu du fichier téléchargé dans un répertoire de votre choix (par exemple, C:\Program Files\Apache\cxf-3.x.x).

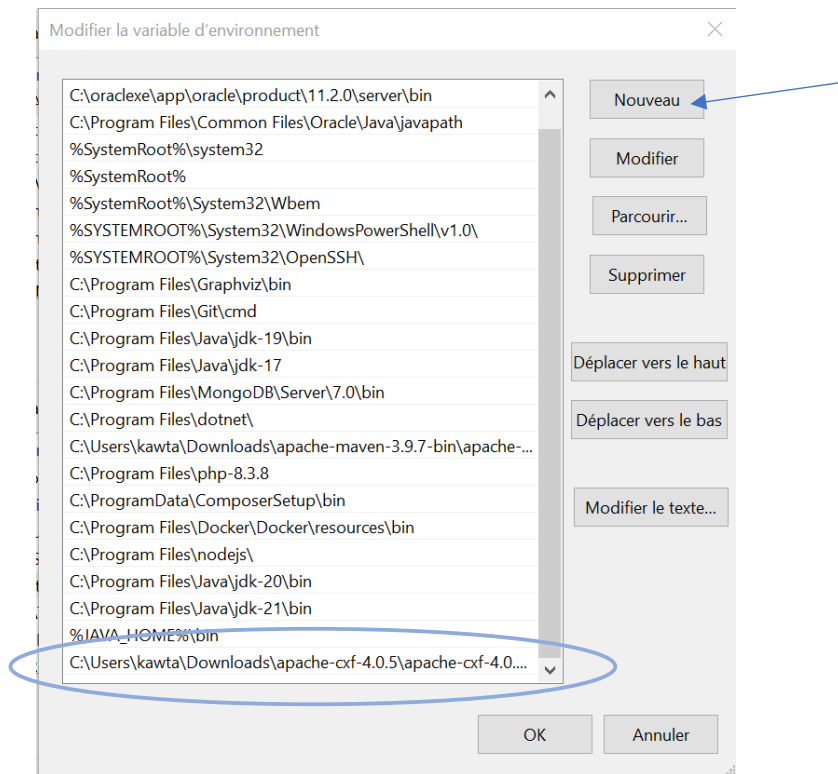
#### 3. Localiser le répertoire bin d'Apache CXF :

- Le répertoire bin contient les scripts nécessaires pour utiliser les outils d'Apache CXF, notamment wsdl2java pour générer un client SOAP à partir d'un WSDL.

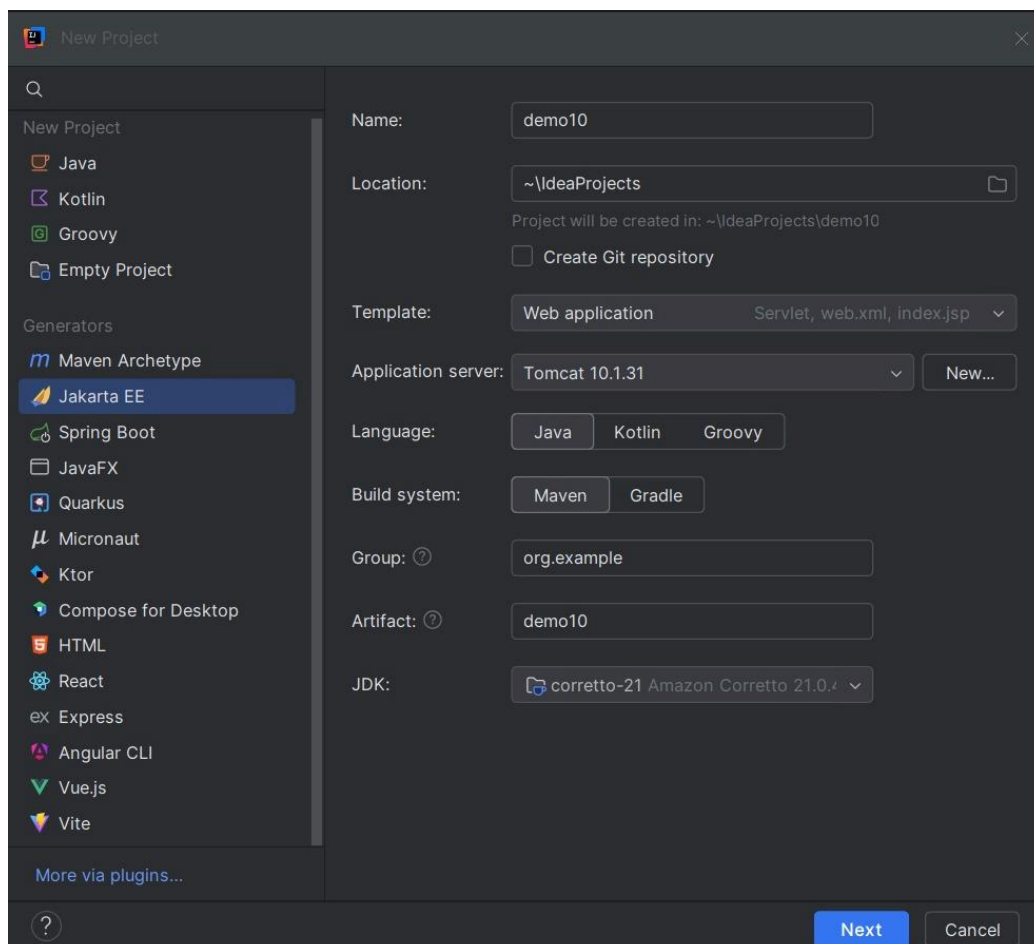
#### 4. Ajouter le chemin aux variables d'environnement :

- Allez dans **Paramètres Système Avancés > Variables d'environnement**.
- Dans **Variables système**, recherchez la variable PATH, cliquez sur **Modifier**, puis ajoutez le chemin du dossier bin d'Apache CXF (par exemple, C:\Program Files\Apache\cxf-3.x.x\bin).





### 3. Création d'un nouveau projet



## 4. Configuration de Maven pour Apache CXF

Dans le fichier **pom.xml** de votre projet, vous devrez ajouter les dépendances suivantes pour utiliser Apache CXF avec la prise en charge des services SOAP et XML.

### Dépendances pour Apache CXF

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-transport-http</artifactId>
  <version>3.5.0</version>
</dependency>
```

- **Description** : Cette dépendance ajoute la prise en charge des transports HTTP pour Apache CXF, permettant aux clients et aux serveurs de communiquer sur le protocole HTTP, qui est le protocole standard pour les services SOAP.

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-front-end-jaxws</artifactId>
  <version>3.5.0</version>
</dependency>
```

- **Description** : Cette dépendance ajoute le support pour la génération de clients SOAP avec CXF utilisant la spécification JAX-WS. Elle contient les classes nécessaires pour créer et utiliser des services web en s'appuyant sur des annotations et conventions JAX-WS.

### Dépendances pour Jakarta XML Binding (JAXB)

```
<dependency>
  <groupId>jakarta.xml.bind</groupId>
  <artifactId>jakarta.xml.bind-api</artifactId>
  <version>3.0.1</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
  <version>3.0.1</version>
</dependency>
```

- **Description** : Ces deux dépendances sont nécessaires pour utiliser JAXB (Jakarta XML Binding), une bibliothèque qui permet de lier les objets Java aux fichiers XML et vice versa. La première dépendance fournit l'API, tandis que la seconde est une implémentation de référence de l'API JAXB pour exécuter ces liaisons XML en temps d'exécution.

### Dépendances pour Jakarta Web Services API (JAX-WS)

```
<dependency>
  <groupId>jakarta.jws</groupId>
  <artifactId>jakarta.jws-api</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
  <groupId>jakarta.xml.ws</groupId>
  <artifactId>jakarta.xml.ws-api</artifactId>
  <version>3.0.0</version>
</dependency>
```

- **Description** : Ces dépendances fournissent l'API pour Jakarta XML Web Services (JAX-WS) et Jakarta Web Services (JWS), qui sont les spécifications pour la création de services web SOAP en Java. Elles définissent les annotations et les interfaces de base pour les services web.

```
<dependency>
  <groupId>com.sun.xml.ws</groupId>
  <artifactId>jaxws-rt</artifactId>
  <version>4.0.0</version>
</dependency>
```

- **Description** : Il s'agit de l'implémentation de référence pour JAX-WS. Elle est essentielle pour exécuter les services JAX-WS en fournissant le runtime nécessaire pour que CXF fonctionne avec ces API.

### Dépendances pour Jakarta SOAP (SAAJ)

```
<dependency>
  <groupId>com.sun.xml.messaging.saa</groupId>
  <artifactId>saa</artifactId>
  <version>3.0.2</version>
</dependency>
<dependency>
  <groupId>jakarta.xml.soap</groupId>
  <artifactId>jakarta.xml.soap-api</artifactId>
  <version>3.0.2</version>
</dependency>
```

- **Description** : Ces dépendances fournissent le support pour SOAP (Simple Object Access Protocol) avec l'API Jakarta SOAP (SAAJ). SAAJ permet de manipuler les messages SOAP en Java, en offrant un accès direct aux parties des messages SOAP. La première dépendance est l'implémentation de l'API, tandis que la seconde définit les classes et interfaces pour SAAJ.

## 5. Génération du Client SOAP avec wsdl2java

Après avoir configuré Maven et les variables d'environnement pour Apache CXF, la prochaine étape est de générer les classes Java pour le client SOAP en utilisant l'outil wsdl2java d'Apache CXF. Cet outil prend en entrée un fichier WSDL et génère des classes Java correspondant aux services et opérations décrits dans le WSDL.

### Commande à exécuter

Utilisez la commande suivante pour générer les classes Java :

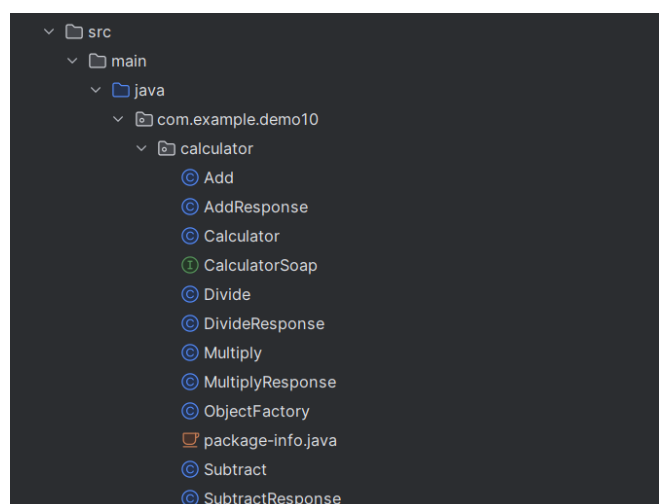
```
wsdl2java.bat -p "com.example.nom_projet.client" -d "src/main/java" http://www.dneonline.com/calculator.asmx?wsdl
```

### Explication des options

- **-p "com.example.nom\_Projet.client"** : spécifie le package dans lequel les classes générées seront placées. (Remplacez monProjet par le nom du projet spécifique).
- **-d "src/main/java"** : indique le répertoire de destination pour les fichiers générés. Ici, les classes seront placées dans src/main/java.
- **"http://www.dneonline.com/calculator.asmx?wsdl"** : l'URL du fichier WSDL du service web SOAP que nous souhaitons consommer.

## 6. Les classes nécessaires générées

Vous pouvez trouver les classes dans le package désiré :





## 7. Exécution des appels SOAP

Une fois les classes Java générées, vous pouvez maintenant utiliser ces classes pour appeler le service SOAP. Voici un exemple d'utilisation du client généré pour invoquer une méthode de ce service avec lequel nous travaillons

```
ClientAPP.java x
4 import com.example.test.client.CalculatorSoap;
5
6 public class ClientAPP {
7     public static void main(String[] args) {
8         // Instantiate the service and port
9         Calculator calculatorService = new Calculator();
10        CalculatorSoap calculatorPort = calculatorService.getCalculatorSoap();
11
12        // Call the Add method
13        int resultAdd = calculatorPort.add(intA: 10, intB: 20);
14        System.out.println("Addition Result: " + resultAdd);
15
16        // Call the Subtract method
17        int resultSubtract = calculatorPort.subtract(intA: 20, intB: 5);
18        System.out.println("Subtraction Result: " + resultSubtract);
19
20        // Call the Multiply method
21        int resultMultiply = calculatorPort.multiply(intA: 3, intB: 4);
22        System.out.println("Multiplication Result: " + resultMultiply);
23
24        // Call the Divide method
25        int resultDivide = calculatorPort.divide(intA: 40, intB: 8);
26        System.out.println("Division Result: " + resultDivide);
27    }
28 }
29
```

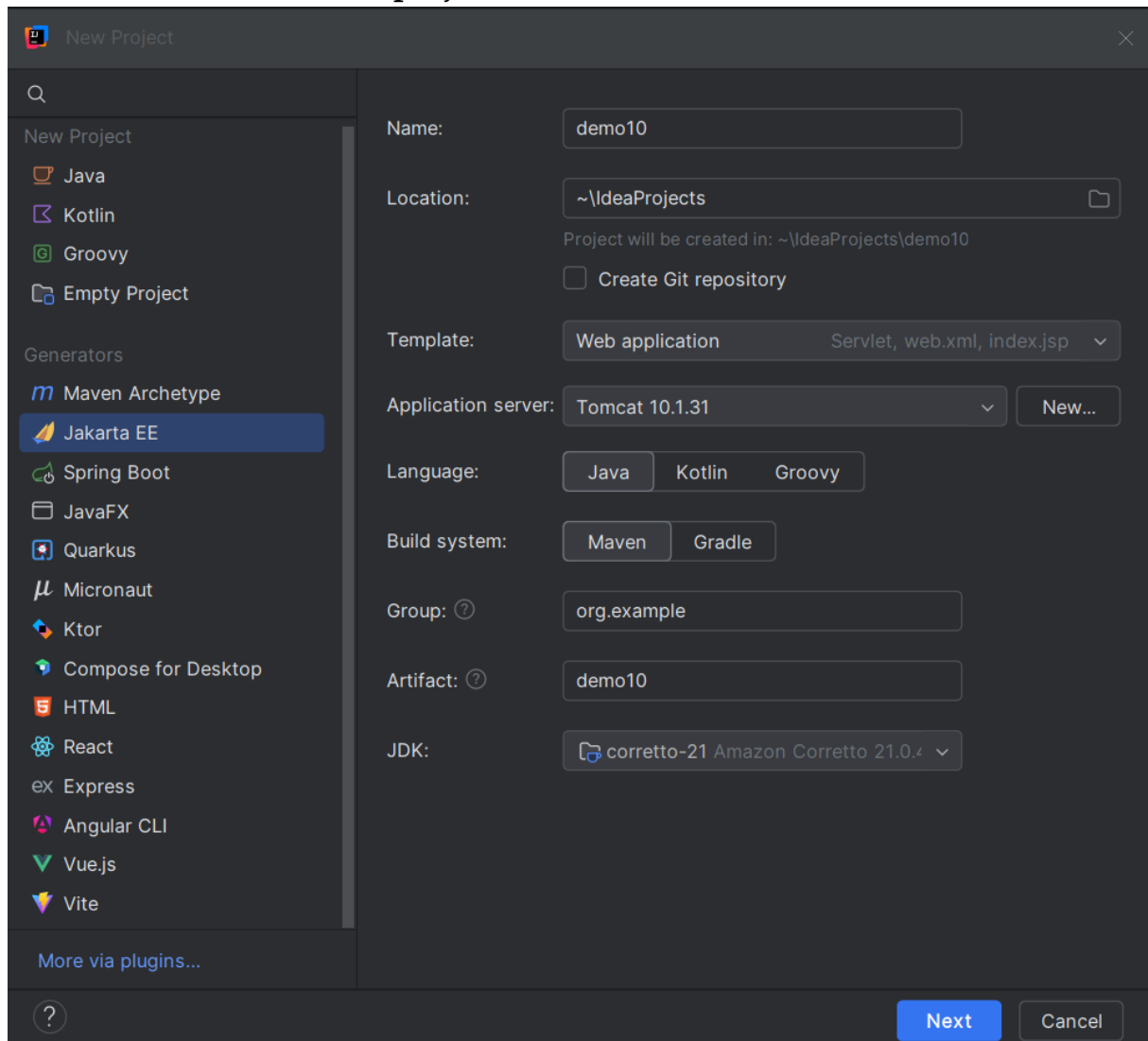
### Résultats

Après avoir exécuté le code ci-dessus, vous devriez obtenir une sortie similaire à celle-ci:

```
Run ClientAPP x
Addition Result: 30
Subtraction Result: 15
Multiplication Result: 12
Division Result: 5
Process finished with exit code 0
```

## Partie 2 : Génération d'un client SOAP avec JAX-WS

### 1. Création d'un nouveau projet



### 2. Configuration des dépendances Maven

Ouvrez le fichier pom.xml et ajoutez les dépendances nécessaires. Le plugin jaxws-maven-plugin sera utilisé pour générer les classes à partir du fichier WSDL.

#### 2.1. Dépendances JAX-WS et JAXB

##### jaxws-api (API de JAX-WS)

```
<dependency>
  <groupId>javax.xml.ws</groupId>
  <artifactId>jaxws-api</artifactId>
  <version>2.3.1</version> <!-- Version stable de JAX-WS -->
</dependency>
```

- **Description :** JAX-WS (Java API for XML Web Services) est une API permettant de créer des services Web SOAP en Java. Cette dépendance contient les interfaces nécessaires (comme Service ou WebServiceContext) pour développer des clients et des services SOAP et ne contient que les interfaces, pas l'implémentation (runtime).

#### jaxws-rt (Runtime JAX-WS, implémentation GlassFish)

```

46      <dependency>
47      <groupId>com.sun.xml.ws</groupId>
48      <artifactId>jaxws-rt</artifactId>
49      <version>2.3.3</version> <!-- Runtime pour JAX-WS -->
50      </dependency>
51

```

- **Description :** Cette dépendance fournit le runtime nécessaire pour exécuter le code basé sur JAX-WS. Elle est maintenue par GlassFish et contient toutes les classes concrètes nécessaires pour le déploiement et l'exécution de services SOAP et elle inclut aussi le tooling comme wsimport et wsgen pour générer des classes Java à partir de WSDL ou vice-versa.

#### jaxb-api (API JAXB pour sérialisation XML)

```

52      <!-- JAXB API pour sérialisation et désérialisation XML -->
53      <dependency>
54      <groupId>javax.xml.bind</groupId>
55      <artifactId>jaxb-api</artifactId>
56      <version>2.3.1</version>
57      </dependency>
58

```

- **Description :** JAXB (Java Architecture for XML Binding) est une API permettant de convertir des objets Java en XML et inversement (marshalling/demarshalling).

#### jaxb-runtime (Runtime JAXB, implémentation GlassFish)

```

59      <!-- JAXB Runtime -->
60      <dependency>
61      <groupId>org.glassfish.jaxb</groupId>
62      <artifactId>jaxb-runtime</artifactId>
63      <version>2.3.3</version>
64      </dependency>
65

```

- **Description :** Cette dépendance fournit l'implémentation de l'API JAXB (runtime) nécessaire à l'exécution du processus de marshalling et de demarshalling XML. Elle est maintenue par GlassFish et inclut toutes les classes concrètes nécessaires à l'exécution de la sérialisation JAXB.

## 2.2. Plugin jaxws-maven-plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>3.4.0</version>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>jaxws-maven-plugin</artifactId>
      <version>2.6</version>
      <executions>
        <execution>
          <goals>
            <goal>wsimport</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <wsdlUrls>
          <wsdlUrl>http://www.dneonline.com/calculator.asmx?wsdl</wsdlUrl> <!-- Remplacez par l'URL de votre WSDL -->
        </wsdlUrls>
        <packageName>com.example.demo10.calculator</packageName>
        <sourceDestDir>src/main/java</sourceDestDir>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- **Explication :**
  - **jaxws-maven-plugin :** Ce plugin est utilisé pour générer les classes Java à partir du fichier WSDL.
  - **wsdlUrl :** L'URL du WSDL du service SOAP que nous exploitons.
  - **packageName :** Le package dans lequel les classes générées seront placées.
- **Instructions :**
  - Le service SOAP que nous exploitons dans ce cas est celui dont le WSDL est accessible depuis l'adresse : <http://www.dneonline.com/calculator.asmx?wsdl>
  - Configurez le packageName selon la structure de votre projet.

## 3. Commande Maven pour la génération des classes

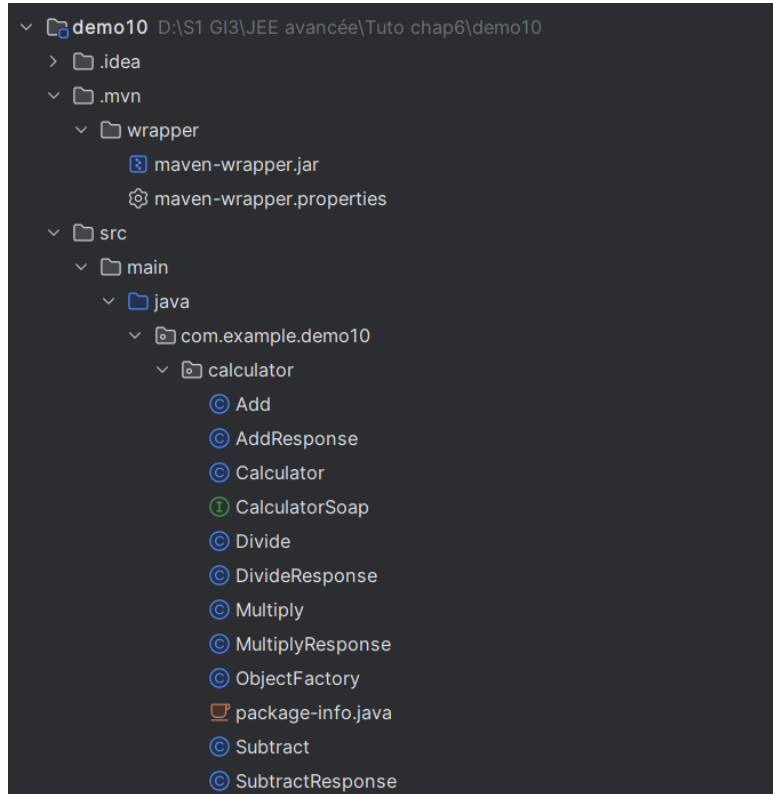
Exécutez sur le terminal Maven la commande : **mvn generate-resources** .

- **Explication :**

Si vous utilisez un plugin comme jaxws-maven-plugin pour générer des classes à partir d'un fichier WSDL, cette génération pourrait se produire lors de la phase generate-resources. Le plugin est configuré pour télécharger et traiter le WSDL afin de créer des stubs Java.

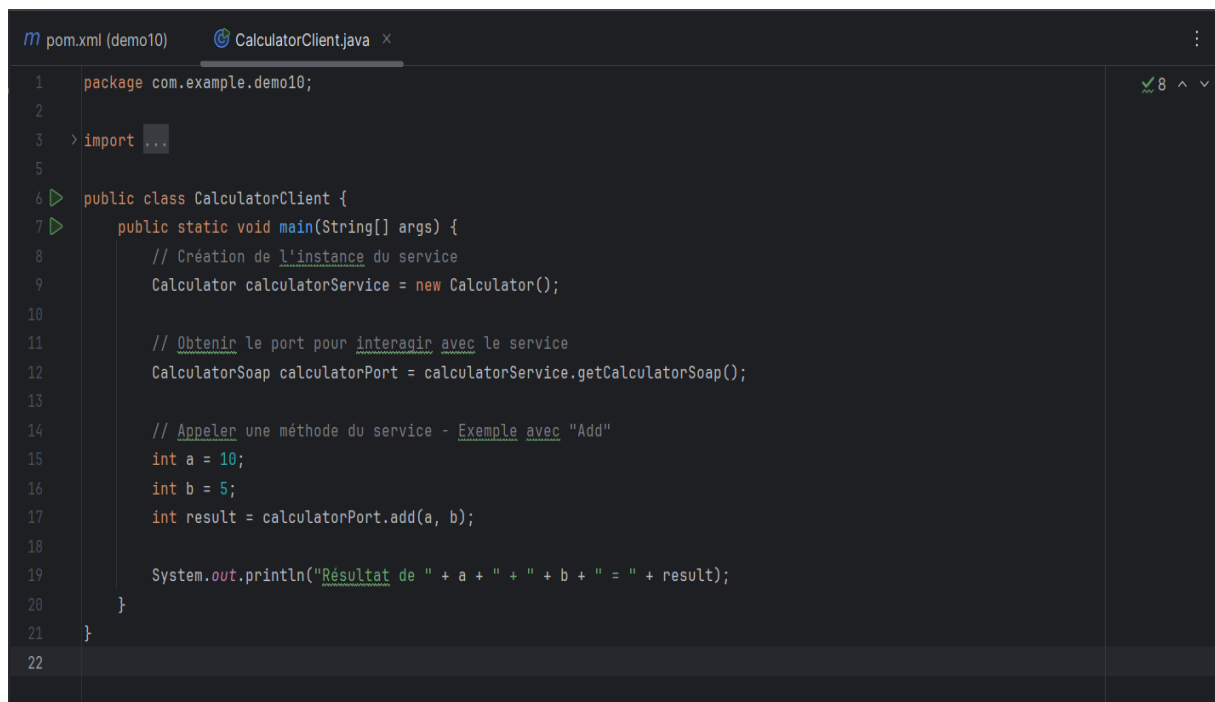
## 4. Les classes nécessaires générées

Vous pouvez trouver les classes dans le package désiré :



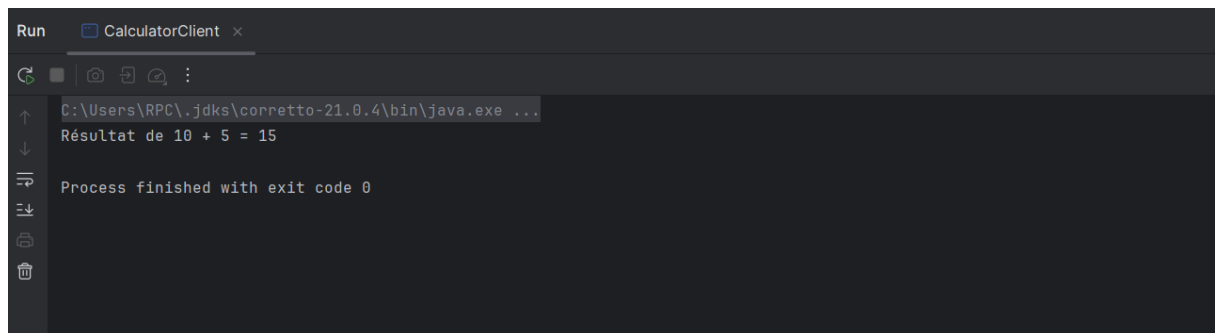
## 5. Création d'une classe client SOAP

Nous allons maintenant créer une classe Java qui utilise les classes générées pour appeler le service SOAP. Le service SOAP fournit des opérations de calcul simple, comme l'addition et la soustraction.



## 6. Les résultats obtenus

Après avoir exécuté le code ci-dessus, vous devriez obtenir une sortie similaire à celle-ci:



```
Run CalculatorClient x
C:\Users\RPC\.jdk\corretto-21.0.4\bin\java.exe ...
Résultat de 10 + 5 = 15
Process finished with exit code 0
```

## Conclusion

En suivant ce tutoriel, vous avez découvert deux approches puissantes pour consommer un service SOAP en Java : **Apache CXF** et **JAX-WS**. Chaque méthode a ses particularités et avantages. Apache CXF se distingue par sa richesse en fonctionnalités et son intégration avec d'autres frameworks, tandis que JAX-WS offre une approche plus native et légère pour ceux qui recherchent une implémentation standard. Grâce à ces deux solutions, vous êtes désormais capable de **générer un client SOAP**, de configurer l'environnement adéquat et d'exécuter des appels vers un service web à partir d'un fichier **WSDL**. Vous pouvez ainsi choisir la méthode qui correspond le mieux aux besoins de vos projets futurs, qu'il s'agisse de simplicité ou de flexibilité accrue.